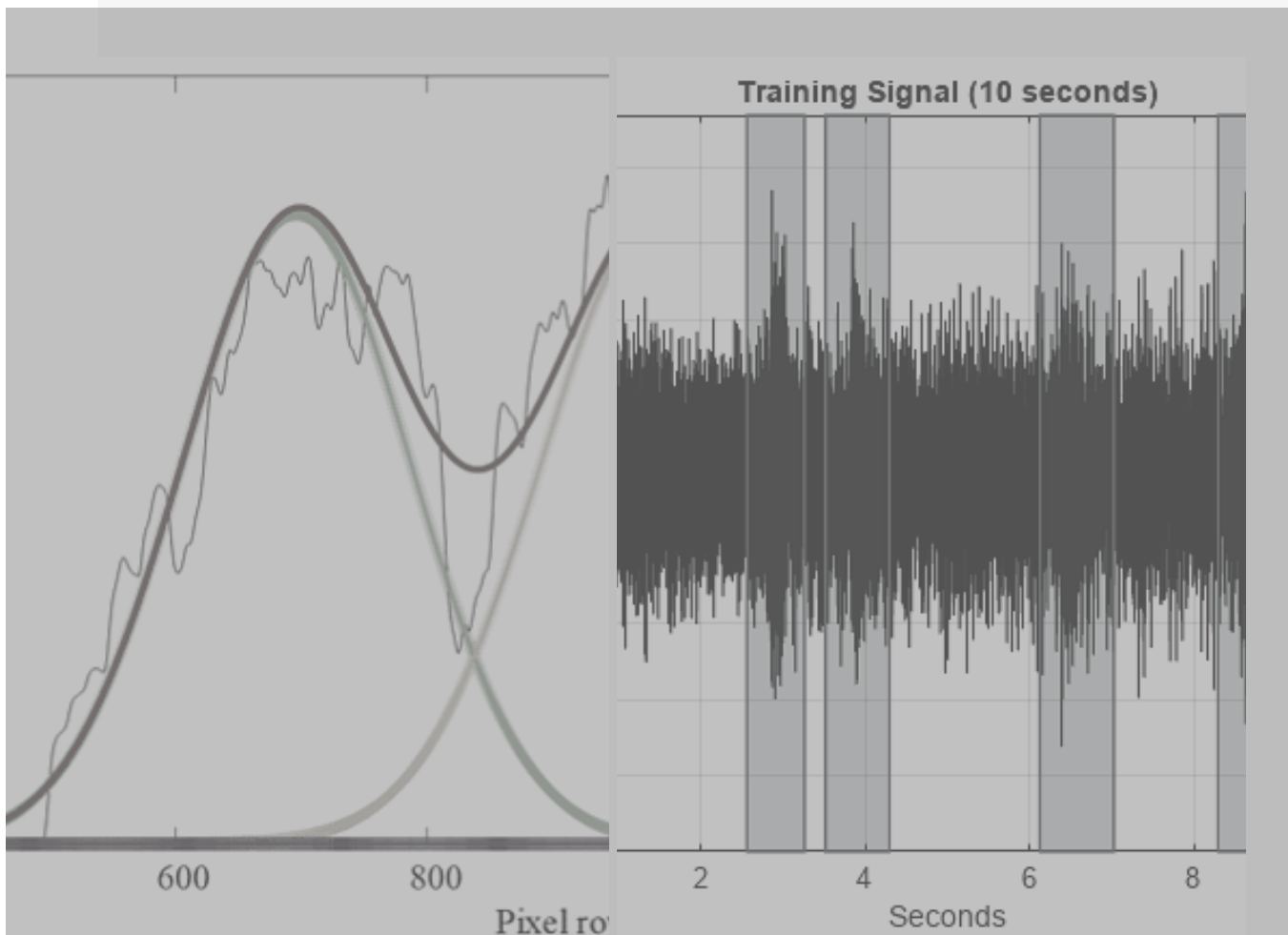


## TRAITEMENT AUTOMATIQUE DE PAROLE

MAI 2023

## DESCRIPTIF DU PROJET

L'IMPLÉMENTATION DU GMM POUR  
LA RECONNAISSANCE DE LANGUE**PRÉPARÉ PAR :**

IIKRAM BELMADANI

**ENCADRÉ PAR :**

PR, KHARROUBI JAMAL

---

# INTRODUCTION :

La reconnaissance automatique de langue fait partie de la reconnaissance automatique de parole qui est un sous-domaine de l'analyse de la parole. C'est l'ensemble de techniques permettant de transformer la parole humaine en données normalisées (texte, intention, valeurs,...) qui peuvent être traitées par les machines. C'est l'une des tâches les plus importantes et les plus difficiles dans le domaine de l'intelligence artificielle et de l'apprentissage automatique.

L'application de la reconnaissance de langue est utilisée en particulier dans le domaine des technologies de l'information et de la communication, où la compréhension de différentes langues est un facteur clé dans la communication et l'apprentissage des interactions interculturelles.

---

# DESCRIPTION DU PROJET

Dans le cadre de ce projet, nous visons à construire un système de reconnaissance de langue notamment des langues suivantes :

- **Arabe**
- **Français**
- **Anglais**
- **Espagnol**
- **Japonais**
- **Russe**

Nous utiliserons un ensemble de données d'enregistrements audio pour former un modèle d'apprentissage automatique capable de reconnaître la langue parlée.

Les étapes que nous avons suivies pour réaliser ce projet sont les suivantes :

- **Etape 1** : Rassembler l'ensemble des données et construire une dataset qui contient assez d'enregistrements audio pour l'entraînement du modèle.
- **Etape 2** : Lecture des enregistrements audio.
- **Etape 3** : Extraction des coefficients cepstraux de fréquence Mel (MFCC) et Prétraitement qui consiste à supprimer le silence

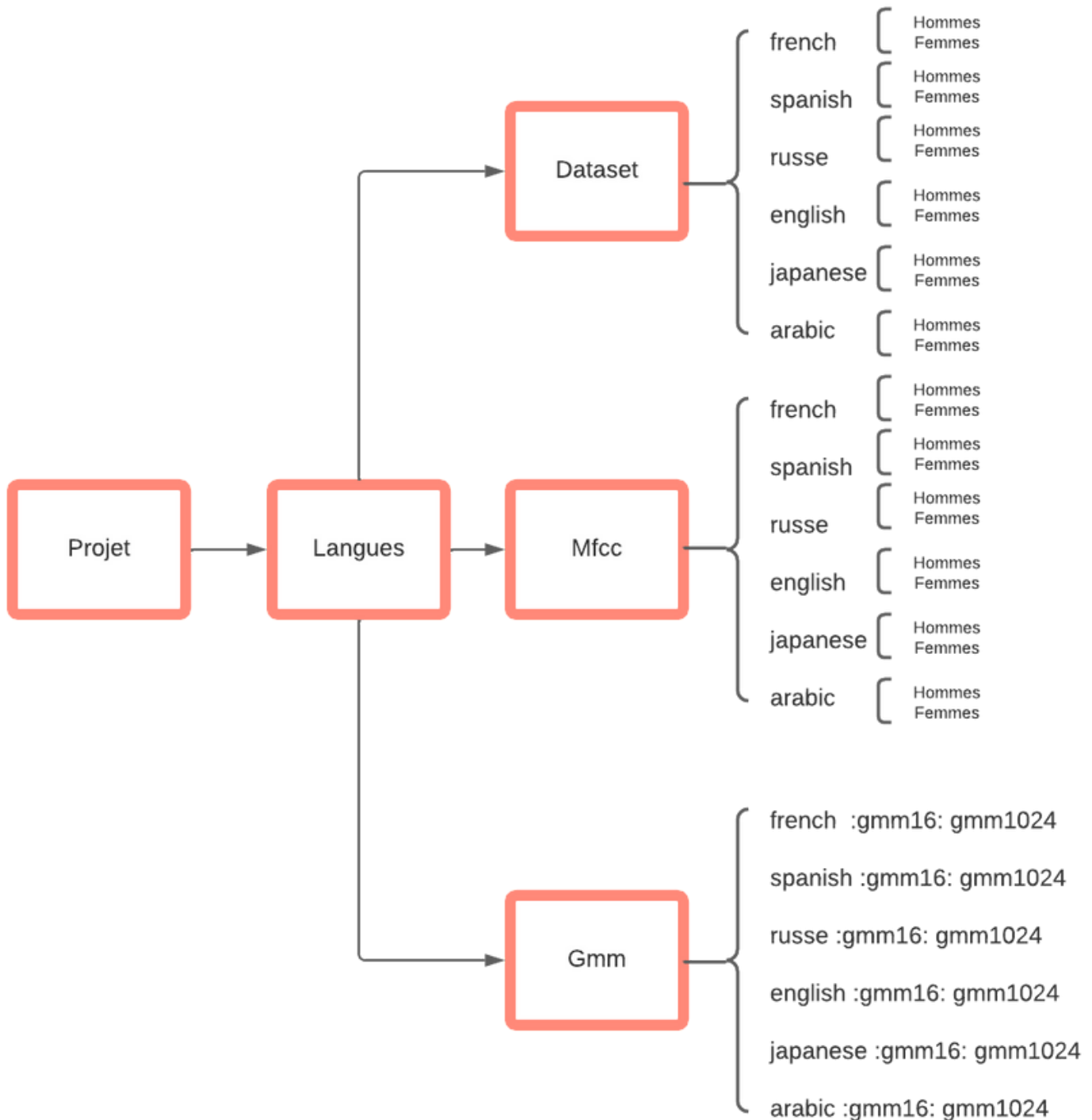
- 
- **Etape 4** : Diviser l'ensemble des données en données d'entraînement et données de test
  - **Etape 5** : Former les modèles GMM pour chaque langue.
  - **Etape 6** : Evaluer les performances de chaque modèle sur l'ensemble des données de test .
  - **Etape 7** : Diviser l'ensemble Test en segments de 5 secondes, 10 s et de 15 s.
  - **Etape 8** : Faire des prédictions sur ces segments et tracer la courbe ROC.

Ce travail est fait avec le langage Python en utilisant l'environnement Jupyter-Notebook. Les modèles de chaque langue sont entraînés dans un notebook où on retrouve les étapes précédentes de 1 à 6. On a donc six notebooks (six langues).

Pour le test, on a aussi six notebooks où on retrouve les étapes 7 et 8.

Dans la suite de ce rapport nous détaillerons chaque étape de la démarche suivie.

# ARBORESCENCE DU PROJET



---

# ETAPE 1 :

## DATASET

La dataset avec laquelle nous travaillons dans ce projet a été construite avec la contribution de tous les étudiants de la classe et notre cher professeur.

Elle est constituée d'enregistrements audio de différentes langues tirés d'internet. Ces enregistrements sont sous format wav.

Les durées de l'ensemble des enregistrements dans chaque langue sont :

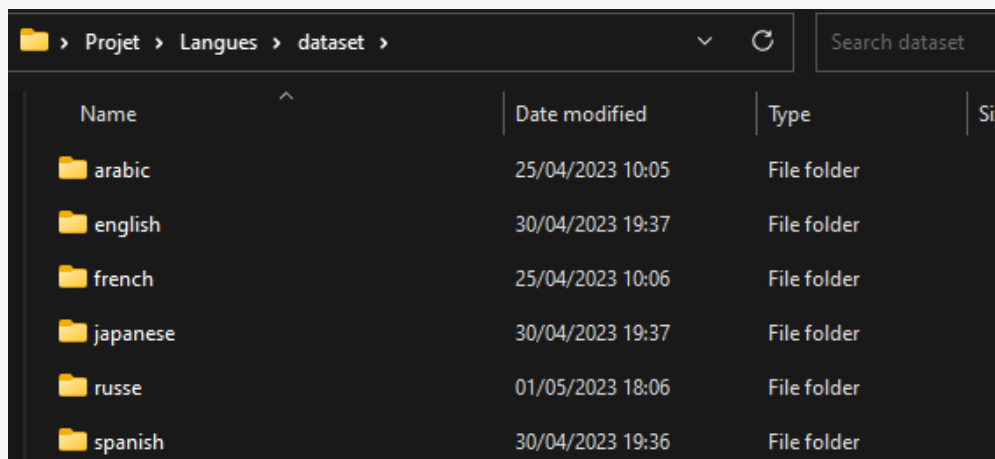
- **Russe** : 57 min
- **Anglais** : 31 min
- **Français** : 32 min
- **Espagnole** : 27.57 min
- **Arabe** : 50,49 min
- **Japonais** : 23,25 min

---

# ETAPE 1 :

## DATASET

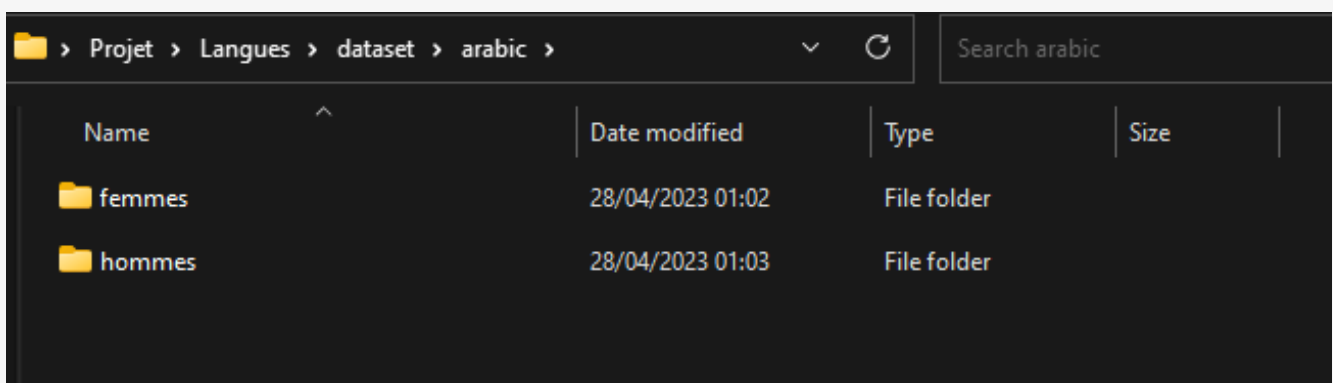
Le dossier Dataset contient six sous-dossiers chacun contient les enregistrements audio de la langue.



Projet > Langues > dataset >			
Name	Date modified	Type	Size
arabic	25/04/2023 10:05	File folder	
english	30/04/2023 19:37	File folder	
french	25/04/2023 10:06	File folder	
japanese	30/04/2023 19:37	File folder	
russe	01/05/2023 18:06	File folder	
spanish	30/04/2023 19:36	File folder	

Chacun de ces six dossiers contient deux sous-dossiers :

- Dossiers Hommes : pour les enregistrements des hommes
- Dossier Femmes : pour les enregistrements des femmes



Projet > Langues > dataset > arabic >			
Name	Date modified	Type	Size
femmes	28/04/2023 01:02	File folder	
hommes	28/04/2023 01:03	File folder	

---

## ETAPE 2 :

### LECTURE DES ENREGISTREMENTS AUDIO

Afin de lire les enregistrements audio nous avons defini la fonction suivante qui permet de lire les audios à partir d'un chemin donné (filepath) à l'aide de la bibliothèque Scipy et retourne trois listes : audios, freqs, filepaths

```
def read_audios(path):
    audios = []
    freqs = []
    filepaths = []
    #walking through the directory that contains the dataset and reading each file that has the .wav extension
    for dp, dn, filenames in os.walk(path):
        for filename in filenames:
            if filename.endswith('.wav'):
                filepath = os.path.join(dp, filename)
                filepaths.append(filepath)
                with open(filepath, "rb") as f:
                    # Load the audio using scipy
                    freq, data = scipy.io.wavfile.read(f, mmap=False)
                    # append the data and frequency to the respective lists
                    audios.append(data)
                    freqs.append(freq)
    return audios, freqs, filepaths
```



---

## ETAPE 3:

### EXTRACTION DES MFCC ET PRÉTRAITEMENT

Après la lecture des enregistrements audio vient l'étape où on doit extraire les coefficients Mfcc et supprimer les trames qui constituent le silence. Pour faire cela, nous avons défini la fonction suivante qui prend en entrée la liste audios, la liste freqs, la liste filepaths, et le chemin où souhaite enregistrer les MFCCs.

```
def extractMfccs_RemoveSilence_saveMfccs(audios, freqs, filepaths, directory):
    mfccs = []

    for audio, freq, filepath in zip(audios, freqs, filepaths):
        # extract the MFCC features
        mfcc_features = mfcc(audio, freq, winlen=0.025, winstep=0.01, numcep=13, nfilt=26, nfft= 2048, lowfreq=0,
                             highfreq=None, preemph=0.97, ceplifter=22, appendEnergy=False)

        # calculate the energy
        energy = np.sum(mfcc_features**2, axis=1)
        # calculate the threshold for silence
        threshold = np.mean(energy) * 0.4
        #removing silence frames from mfccs
        voiced_indices = np.where(energy > threshold)[0]
        mfccs_voiced = mfcc_features[voiced_indices,:]
        mfccs.append(mfccs_voiced)

        # print the shape of the MFCCs before and after removing silence
        print(f"MFCCs before removing silence: {mfcc_features.shape}")
        print(f"MFCCs after removing silence: {mfccs_voiced.shape}")

    #saving mffcs
    # extract the gender information from the file name
    gender = None
    if 'Hommes' in filepath:
        gender = 'Hommes'
    elif 'Femmes' in filepath:
        gender = 'Femmes'

    # save the MFCCs to the appropriate directory based on gender
    if gender is not None:
        gender_dir = os.path.join(directory, gender)
        if not os.path.exists(gender_dir):
            os.makedirs(gender_dir)
        mfcc_file = os.path.join(gender_dir, os.path.splitext(os.path.basename(filepath))[0] + ".mfcc")
        np.savetxt(mfcc_file, mfccs_voiced, delimiter=',')

    return mfccs
```

---

## ETAPE 3:

### EXTRACTION DES MFCC ET PRÉTRAITEMENT

- Afin de supprimer le silence, nous avons calculé l'énergie du signal vocal représenté sous forme MFCC. Elle est calculée pour chaque trame des MFCC à l'aide de la bibliothèque numpy. Après avoir calculé l'énergie, un seuil est calculé à 40 % de l'énergie moyenne. Ce seuil est utilisé pour distinguer les trames du silence des trames de parole. Les trames de parole correspondent aux trames où l'énergie est supérieure au seuil, tandis que les trames du silence correspondent aux trames où l'énergie est inférieure.
- Les MFCC extraites sont enregistrées dans des fichiers basés sur le genre. La fonction extrait d'abord les informations de genre du nom du fichier en vérifiant s'il contient la chaîne "Hommes" ou "Femmes". Ensuite, elle enregistre les MFCC dans un répertoire nommé d'après le genre, et si le répertoire n'existe pas, il en crée un nouveau. Enfin, elle enregistre les MFCC dans un fichier texte portant le même nom que le fichier audio d'origine, mais avec une extension ".mfcc". Les fonctions MFCC sont enregistrées sous forme de valeurs séparées par des virgules dans le fichier texte.

---

## ETAPE 3:

### EXTRACTION DES MFCC ET PRÉTRAITEMENT

- On peut voir ici que le nombre de trames a diminué après la suppression du silence

```
MFCCs before removing silence: (6423, 13)
MFCCs after removing silence: (6176, 13)
MFCCs before removing silence: (1469, 13)
MFCCs after removing silence: (1444, 13)
MFCCs before removing silence: (2384, 13)
MFCCs after removing silence: (2353, 13)
MFCCs before removing silence: (2836, 13)
MFCCs after removing silence: (2781, 13)
MFCCs before removing silence: (2310, 13)
MFCCs after removing silence: (2231, 13)
MFCCs before removing silence: (3378, 13)
MFCCs after removing silence: (3378, 13)
MFCCs before removing silence: (2768, 13)
MFCCs after removing silence: (2659, 13)
MFCCs before removing silence: (1548, 13)
MFCCs after removing silence: (1514, 13)
MFCCs before removing silence: (3291, 13)
MFCCs after removing silence: (3205, 13)
```

---

## ETAPE 4:

### TRAIN - TEST SPLIT

- Après avoir nettoyé nos données, nous avons défini la fonction suivante qui permet de diviser les MFFCs extraits en lot d'entraînement et lot de test. Afin que le genre ne soit pas considéré comme caractéristique de la langue, nous avons divisés équitablement les MFFCs selon le genre (  $\frac{2}{3}$  des MFFCs Hommes et  $\frac{2}{3}$  des MFFCs Femmes pour l'entraînement et les  $\frac{1}{3}$  qui restent pour le test ).

---

# ETAPE 4:

## TRAIN - TEST SPLIT

---

```
def train_test_split(mfcc_dir):
    # create separate lists for male and female file paths
    male_files = []
    female_files = []
    for root, dirs, files in os.walk(mfcc_dir):
        for file in files:
            if file.endswith('.mfcc'):
                if 'Hommes' in root:
                    male_files.append(os.path.join(root, file))
                elif 'Femmes' in root:
                    female_files.append(os.path.join(root, file))

    # shuffle the male and female lists independently
    random.shuffle(male_files)
    random.shuffle(female_files)

    # split the male and female lists into train and test based on the desired ratio
    male_train = male_files[:int(2/3*len(male_files))]
    male_test = male_files[int(2/3*len(male_files)):]

    female_train = female_files[:int(2/3*len(female_files))]
    female_test = female_files[int(2/3*len(female_files)):]

    # merge the train and test lists for both male and female
    train_files = male_train + female_train
    test_files = male_test + female_test

    # Load the MFCC features from the saved files for the train and test sets
    train_mfccs = []
    test_mfccs = []

    for file in train_files:
        train_mfccs.append(np.loadtxt(file, delimiter=','))

    for file in test_files:
        test_mfccs.append(np.loadtxt(file, delimiter=','))

    # print the shapes of the train and test MFCC feature arrays
    print(f"Train male MFCCs shape: {np.array(male_train).shape}")
    print(f"Test male MFCCs shape: {np.array(male_test).shape}")
    print(f"Train female MFCCs shape: {np.array(female_train).shape}")
    print(f"Test female MFCCs shape: {np.array(female_test).shape}")
    print(f"Train MFCCs shape: {np.array(train_mfccs).shape}")
    print(f"Test MFCCs shape: {np.array(test_mfccs).shape}")

    return train_mfccs, test_mfccs
```

---

## ETAPE 5:

### CONSTRUCTION DES GMMs

- Dans cette étape nous avons défini sept fonctions pour entraîner les différents modèles GMMs. Ce qui change dans ces fonctions est le nombre de gaussiennes considérées (16 – 32 – 64 – 128 – 256 – 512 – 1024 ). Ces fonctions permettent d'initialiser un modèle GMM, l'entraîner puis l'enregistrer dans un fichier pickle.

```
def gmm16(train_mfccs):  
    # Initialize the GMM model with 16 classes  
    gmm = GaussianMixture(n_components=16, covariance_type='diag', random_state=0)  
  
    # Fit the GMM model to the training data  
    gmm.fit(train_mfccs)  
  
    # Save the trained GMM model to a file  
    joblib.dump(gmm, r'C:\Users\ASUS ROG STRIX\Desktop\Projet\Langues\gmm\russe\gmm_model16_russe.pkl')  
  
    return gmm
```

```
def gmm32(train_mfccs):  
    # Initialize the GMM model with 32 classes  
    gmm = GaussianMixture(n_components=32, covariance_type='diag', random_state=0)  
  
    # Fit the GMM model to the training data  
    gmm.fit(train_mfccs)  
  
    # Save the trained GMM model to a file  
    joblib.dump(gmm, r'C:\Users\ASUS ROG STRIX\Desktop\Projet\Langues\gmm\russe\gmm_model32_russe.pkl')  
  
    return gmm
```

---

## ETAPE 6:

### Evaluation des performances des modèles

- Afin d'évaluer les performances de chaque modèle GMM, nous avons utilisé la fonction `score_samples()` de Scikit-Learn qui renvoie un array contenant la log-vraisemblance de chaque trame.

```
scores = []
for model in [gmm16, gmm32, gmm64, gmm128, gmm256, gmm512, gmm1024]:
    score = model.score_samples(mfcc_test)
    scores.append(score)

# Print the scores
print('GMM16 score:', scores[0])
print('GMM32 score:', scores[1])
print('GMM64 score:', scores[2])
print('GMM128 score:', scores[3])
print('GMM256 score:', scores[4])
print('GMM512 score:', scores[5])
print('GMM1024 score:', scores[6])
```

---

## ETAPE 6:

### Evaluation des performances des modèles

- Après nous avons calculé la moyenne de ses scores pour chaque modèle, le modèle le plus performant est celui qui a le score le plus proche de 0.

```
#calculating the score of the hole test set  
print('GMM16 score:', scores[0].mean())  
print('GMM32 score:', scores[1].mean())  
print('GMM64 score:', scores[2].mean())  
print('GMM128 score:', scores[3].mean())  
print('GMM256 score:', scores[4].mean())  
print('GMM512 score:', scores[5].mean())  
print('GMM1024 score:', scores[6].mean())
```

Nous avons obtenu les résultats suivant :

- Le modèle le plus performant de la langue russe : GMM512
- Le modèle le plus performant de la langue arabe : GMM256
- Le modèle le plus performant de la langue française : GMM512
- Le modèle le plus performant de la langue japonaise : GMM512
- Le modèle le plus performant de la langue espagnole : GMM512
- Le modèle le plus performant de la langue anglaise : GMM1024



---

## ETAPE 7:

### Division en segments de 5s, 10s et 15s

- Pour diviser l'ensemble des MFCCs de test en segments de 5s, 10s et 15s, nous avons assumé qu'une seconde nous donne 100 frames. La fonction suivante permet de faire cette division :

---

```
def split_audio_test(test_mfccs, segment_length_sec):  
  
    # Compute the number of frames per segment  
    frames_per_sec = 100 # Assuming 100 frames per second  
    frames_per_segment = int(segment_length_sec * frames_per_sec)  
  
    # Split the test audio into segments  
    num_segments = math.ceil(len(test_mfccs) / frames_per_segment)  
    test_segments = []  
    for i in range(num_segments):  
        start_frame = i * frames_per_segment  
        end_frame = min(start_frame + frames_per_segment, len(test_mfccs))  
        segment = test_mfccs[start_frame:end_frame]  
        test_segments.append(segment)  
  
    return test_segments
```

---

## ETAPE 7:

### Division en segments de 5s, 10s et 15s

- Cette division est faite afin d'illustrer l'influence de la durée du segment sur la capacité du modèle à prédire la langue.
- Pour la plupart des langues nous avons obtenus les meilleurs résultats pour les segments de 15s ce qui est logique puisque plus la durée de l'audio est longue, plus il y aura de variations dans la parole qui seront capturées. Cela peut aider les modèles à mieux comprendre le contexte et à améliorer la précision des prédictions.

---

## ETAPE 8:

### Nouvelles prédictions

Pour faire la prediction sur de nouveaux enregistrements nous avons defini la fonction suivante qui prend en entrée :

1. Les coefficients MFCCs.
2. Les modèles GMM pré-entraînés pour chaque langue: `russian_gmm`, `arabic_gmm`, `spanish_gmm`, `english_gmm`, `japanese_gmm`, `french_gmm`.

et retourne la langue prédite pour le fichier audio ainsi que le score associé en calculant les scores de log-vraisemblance à l'aide de la méthode `score_samples()`. Le score le plus élevé est celui de la langue prédite.

---

## ETAPE 8:

### Nouvelles prédictions

```
def predict_language(mfccs, russian_gmm, arabic_gmm, spanish_gmm, english_gmm, japanese_gmm, french_gmm):  
  
    # calculate the scores for each language  
    russian_score = russian_gmm.score_samples(mfccs)  
    arabic_score = arabic_gmm.score_samples(mfccs)  
    spanish_score = spanish_gmm.score_samples(mfccs)  
    english_score = english_gmm.score_samples(mfccs)  
    japanese_score = japanese_gmm.score_samples(mfccs)  
    french_score = french_gmm.score_samples(mfccs)  
  
    # return the language with the highest score  
    scores = {"russian": russian_score.mean(), "arabic": arabic_score.mean(), "spanish": spanish_score.mean(),  
             "english": english_score.mean(), "japanese": japanese_score.mean(), "french": french_score.mean()}  
    predicted_language = max(scores, key=scores.get)  
    max_score = scores[predicted_language]  
  
    return max_score , predicted_language
```

## COURBE ROC

La courbe ROC (Receiver Operating Characteristic) est une courbe qui représente la performance d'un modèle de classification binaire à différents seuils de décision.

Plus précisément, la courbe ROC trace le taux de vrais positifs (TPR) en fonction du taux de faux positifs (FPR) pour différents seuils de décision. Le TPR est le taux de cas positifs correctement identifiés, tandis que le FPR est le taux de cas négatifs incorrectement identifiés comme positifs. Le seuil de décision correspond à la valeur de la probabilité prédite à partir du modèle qui est utilisée pour déterminer la classe de prédiction finale.

Le bloc de code suivant permet de tracer trois courbes ROC:

- ROC pour les segments de 5s
- ROC pour les segments de 10s
- ROC pour les segments de 15s

---

# ETAPE 8:

## COURBE ROC

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Get the predicted scores and true labels for russian with different thresholds
scores5s = [result["maxScore"] for result in results5s]
labels5s = [1 if result["predictedLanguage"] == "russian" else 0 for result in results5s]

scores10s = [result["maxScore"] for result in results10s]
labels10s = [1 if result["predictedLanguage"] == "russian" else 0 for result in results10s]

scores15s = [result["maxScore"] for result in results15s]
labels15s = [1 if result["predictedLanguage"] == "russian" else 0 for result in results15s]

# Compute the FPR, TPR, and AUC for each ROC curve
fpr5s, tpr5s, thresholds5s = roc_curve(labels5s, scores5s, pos_label=1)
roc_auc5s = auc(fpr5s, tpr5s)

fpr10s, tpr10s, thresholds10s = roc_curve(labels10s, scores10s, pos_label=1)
roc_auc10s = auc(fpr10s, tpr10s)

fpr15s, tpr15s, thresholds15s = roc_curve(labels15s, scores15s, pos_label=1)
roc_auc15s = auc(fpr15s, tpr15s)

# Plot the ROC curves on the same figure
plt.figure()
plt.plot(fpr5s, tpr5s, color='darkorange', lw=2, label='ROC curve with threshold=5s (area = %0.2f)' % roc_auc5s)
plt.plot(fpr10s, tpr10s, color='green', lw=2, label='ROC curve with threshold=10s (area = %0.2f)' % roc_auc10s)
plt.plot(fpr15s, tpr15s, color='blue', lw=2, label='ROC curve with threshold=15s (area = %0.2f)' % roc_auc15s)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) for russian language detection')
plt.legend(loc="lower right")
plt.show()
```

# RÉSULTATS OBTENUS :

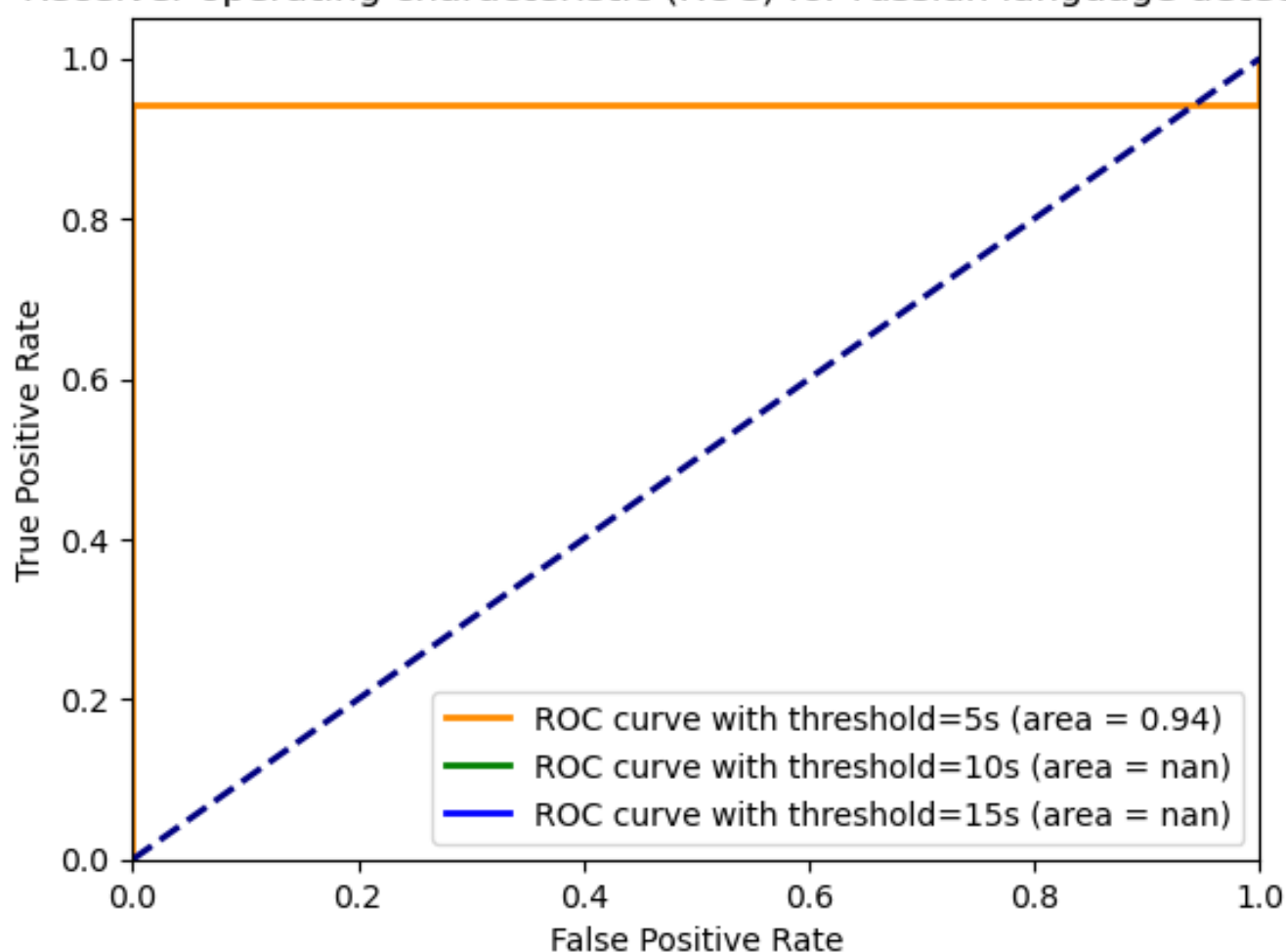
## Pour la langue Russe :

Nous avons obtenu de très bon résultats pour la langue russe en utilisant le model gmm512. En effet pour les segments de 5 s le taux d'erreur est de 6% et pour les segments de 10s et 15s on a obtenu 0 erreur, tous les segments ont été bien prédits.

La courbe Roc résume ces résultats obtenus

(Dans ce cas on ne peut pas avoir les courbes pour les segments de 5s et 10s puisqu'on a juste une seule classe prédite qui est russe donc on ne peut pas mesurer le taux de faux positifs).

Receiver operating characteristic (ROC) for russian language detectio



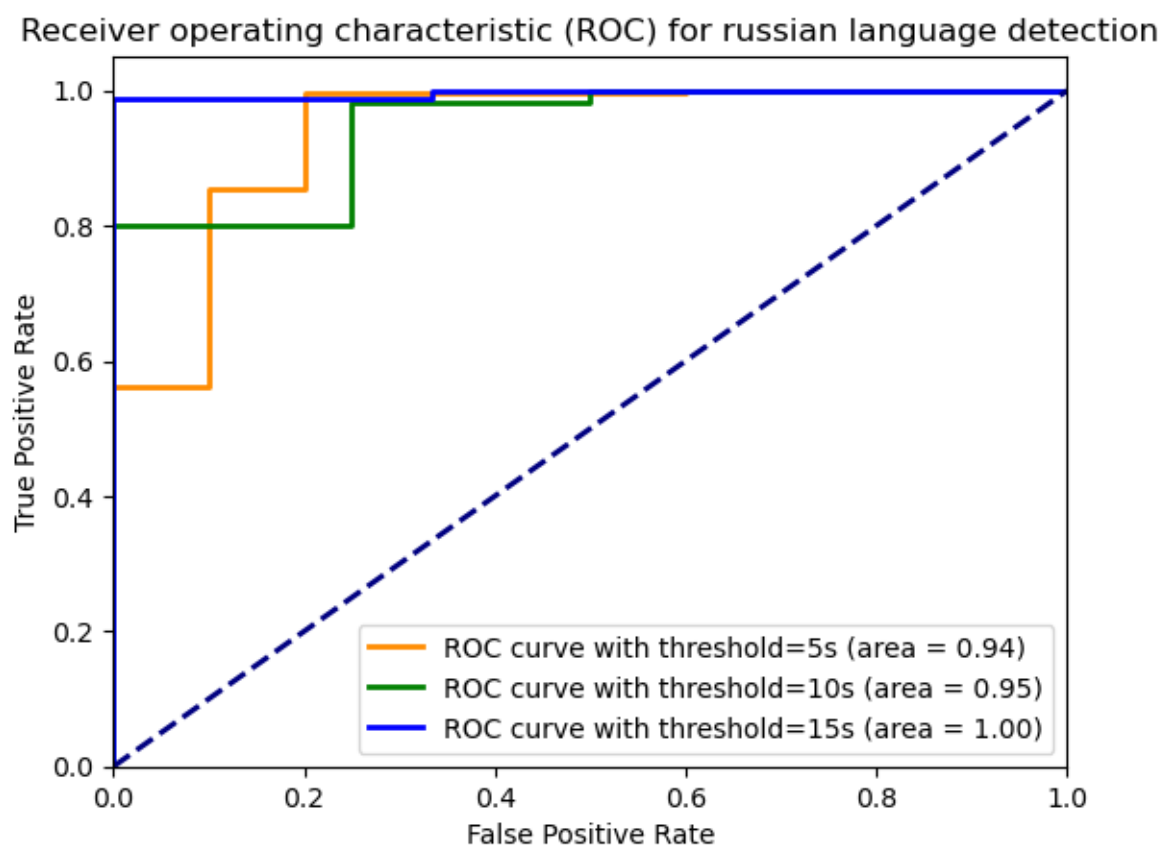
# RÉSULTATS OBTENUS :

## Pour la langue Russe :

Les segments de 5s mal classifiés ont été prédits comme étant des segments de la langue japonaise.

```
MAX_Score: -55.7509554843067 --- Predicted Language: russian  
MAX_Score: -56.43270744342622 --- Predicted Language: russian  
MAX_Score: -55.746567980761604 --- Predicted Language: russian  
MAX_Score: -53.07786915514861 --- Predicted Language: japanese  
MAX_Score: -51.5246014334005 --- Predicted Language: russian  
MAX_Score: -51.00440013886955 --- Predicted Language: russian
```

Si on test avec le modèle gmm16 on obtient des résultats moins précis qu'en utilisant le modèle gmm512

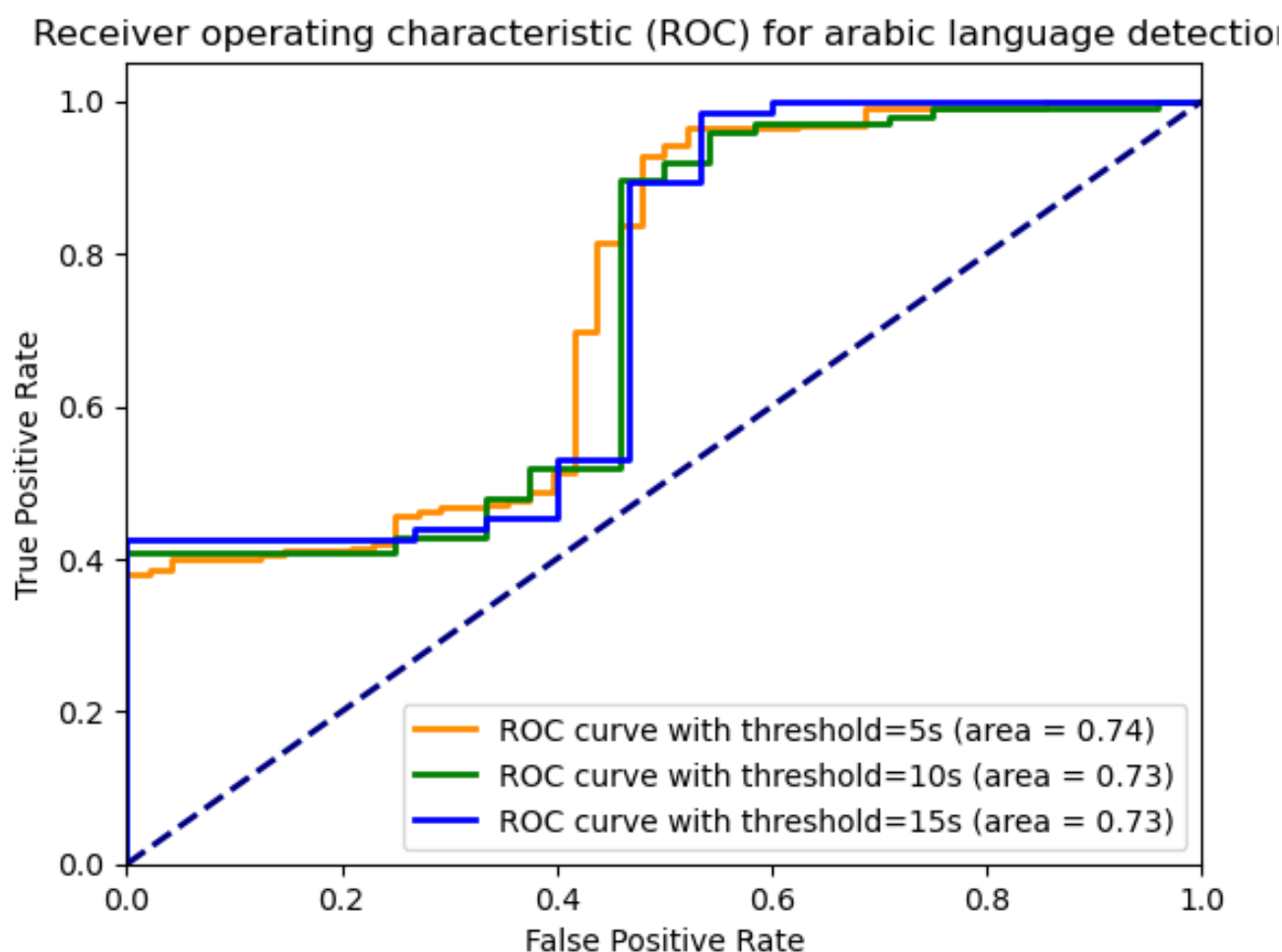


# RÉSULTATS OBTENUS :

## Pour la langue Arabe :

Nous avons obtenu de bon résultats pour la langue arabe en utilisant le model gmm512. En effet pour les segments de 5 s le taux d'erreur de 26% et pour les segments de 10s et 15s on a obtenu un taux d'erreur de 27%.

La courbe Roc résume ces résultats obtenus .





---

# RÉSULTATS OBTENUS :

## Pour la langue arabe :

Les segments mal classifiés ont été prédits comme étant des segments soit de la langue anglaise, française ou bien espagnole .

```
MAX_Score: -54.95358350173258 --- Predicted Language: french
MAX_Score: -56.67944612043838 --- Predicted Language: french
MAX_Score: -54.61432475236154 --- Predicted Language: arabic
MAX_Score: -53.811521354076525 --- Predicted Language: arabic
MAX_Score: -54.27195576975648 --- Predicted Language: arabic
MAX_Score: -54.61613427727946 --- Predicted Language: arabic
MAX_Score: -54.065975340959696 --- Predicted Language: arabic
MAX_Score: -54.36313472609702 --- Predicted Language: arabic
MAX_Score: -54.89376283287489 --- Predicted Language: arabic
MAX_Score: -54.62925696312444 --- Predicted Language: arabic
MAX_Score: -55.22183361153538 --- Predicted Language: arabic
MAX_Score: -55.06388682040211 --- Predicted Language: english
MAX_Score: -54.60865831275721 --- Predicted Language: arabic
MAX_Score: -54.95783204124158 --- Predicted Language: english
MAX_Score: -54.3582336604938 --- Predicted Language: english

MAX_Score: -51.27407212490500 --- Predicted Language: spanish
MAX_Score: -51.75857126098123 --- Predicted Language: spanish
MAX_Score: -52.30684546094028 --- Predicted Language: arabic
MAX_Score: -52.986487994988344 --- Predicted Language: arabic
MAX_Score: -53.08246212071251 --- Predicted Language: arabic
MAX_Score: -54.10345164387472 --- Predicted Language: arabic
MAX_Score: -53.123760031667864 --- Predicted Language: arabic
```

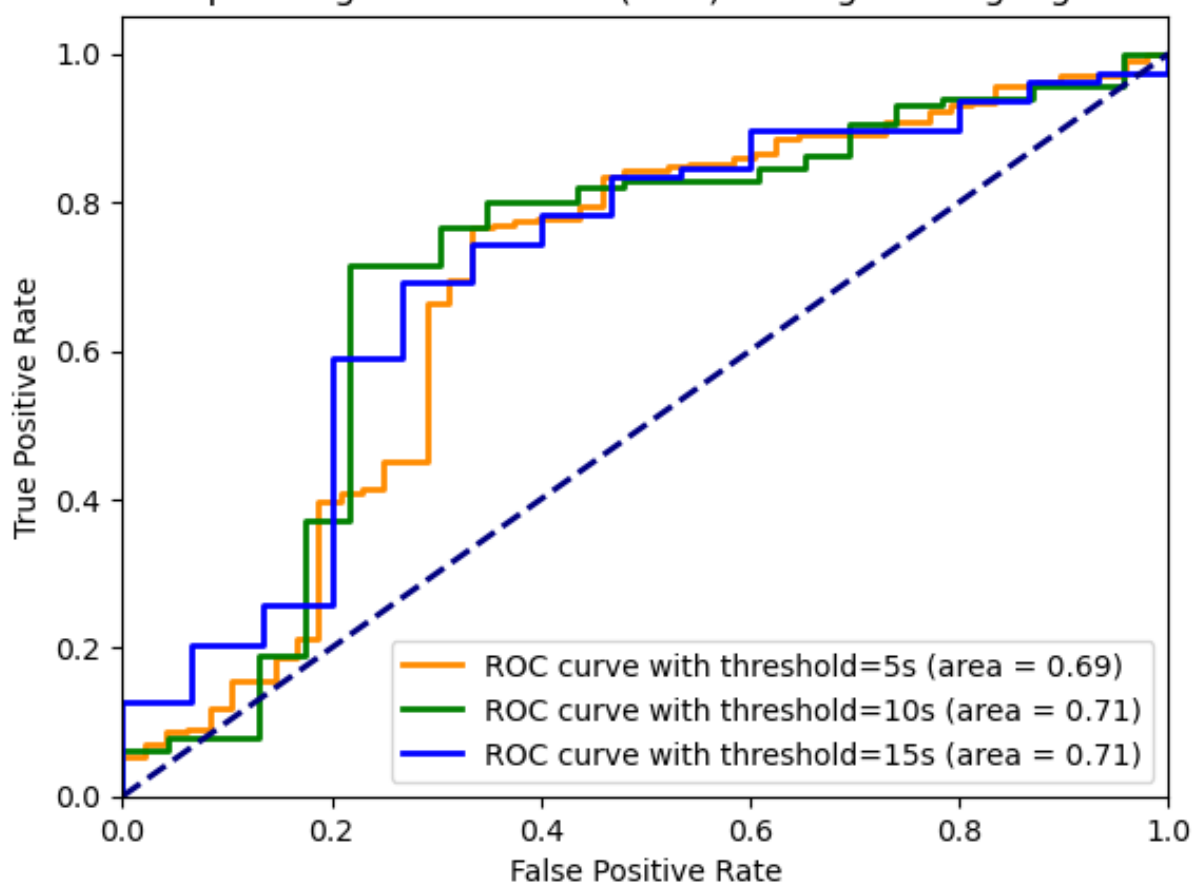
# RÉSULTATS OBTENUS :

## Pour la langue Anglaise :

Nous avons obtenu des résultats plus au moins bons pour la langue anglaise en utilisant le model gmm512. En effet pour les segments de 5 s le taux d'erreur de 31% et pour les segments de 10s et 15s on a obtenu un taux d'erreur de 29%

La courbe Roc résume ces résultats obtenus .

Receiver operating characteristic (ROC) for english language detection



---

# RÉSULTATS OBTENUS :

## Pour la langue anglaise :

Les segments mal classifiés ont été prédits comme étant des segments soit de la langue japonaise, française ou bien russe .

```
MAX_Score: -51.442708919721426 --- Predicted Language: french  
MAX_Score: -52.980107037809354 --- Predicted Language: english  
MAX_Score: -51.56211857743565 --- Predicted Language: english  
MAX_Score: -48.56960618255007 --- Predicted Language: english  
MAX_Score: -52.00795049551125 --- Predicted Language: english  
MAX_Score: -49.31683365101379 --- Predicted Language: english
```

```
MAX_Score: -50.05100552371327 --- Predicted Language: french
```

```
MAX_Score: -53.27961822378109 --- Predicted Language: russian
```

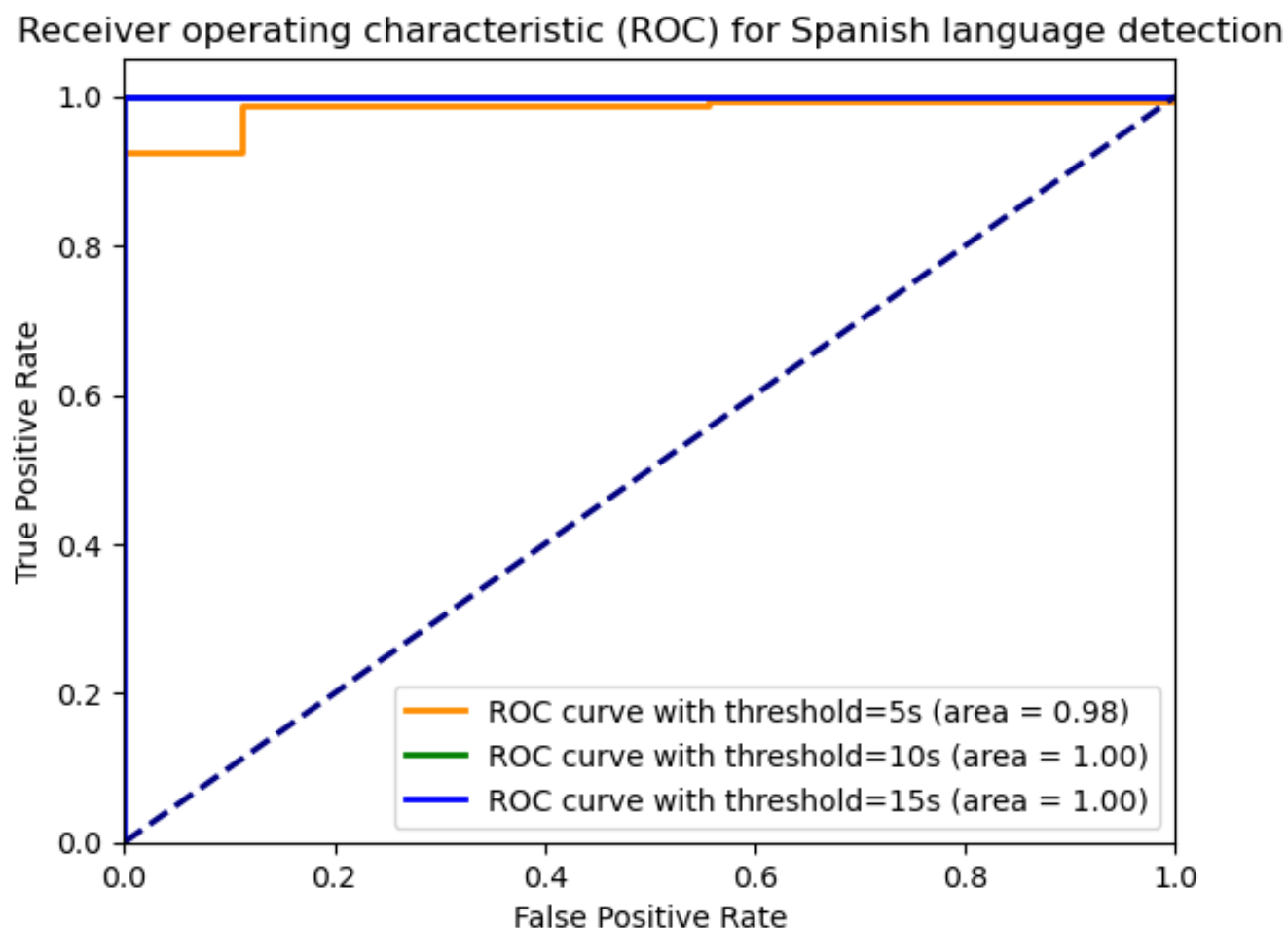
```
MAX_Score: -53.28709063494159 --- Predicted Language: japanese
```

---

# RÉSULTATS OBTENUS :

## Pour la langue Espagnole :

Nous avons obtenu de très bons résultats pour la langue espagnole en utilisant le modèle gmm512. En effet pour les segments de 5 s l'auc est de 0.98 et pour les segments de 10s et 15s l'auc est 1. La courbe Roc résume ces résultats obtenus .



---

# RÉSULTATS OBTENUS :

## Pour la langue espagnole :

Les segments mal classifiés ont été prédits comme étant des segments de la langue française .

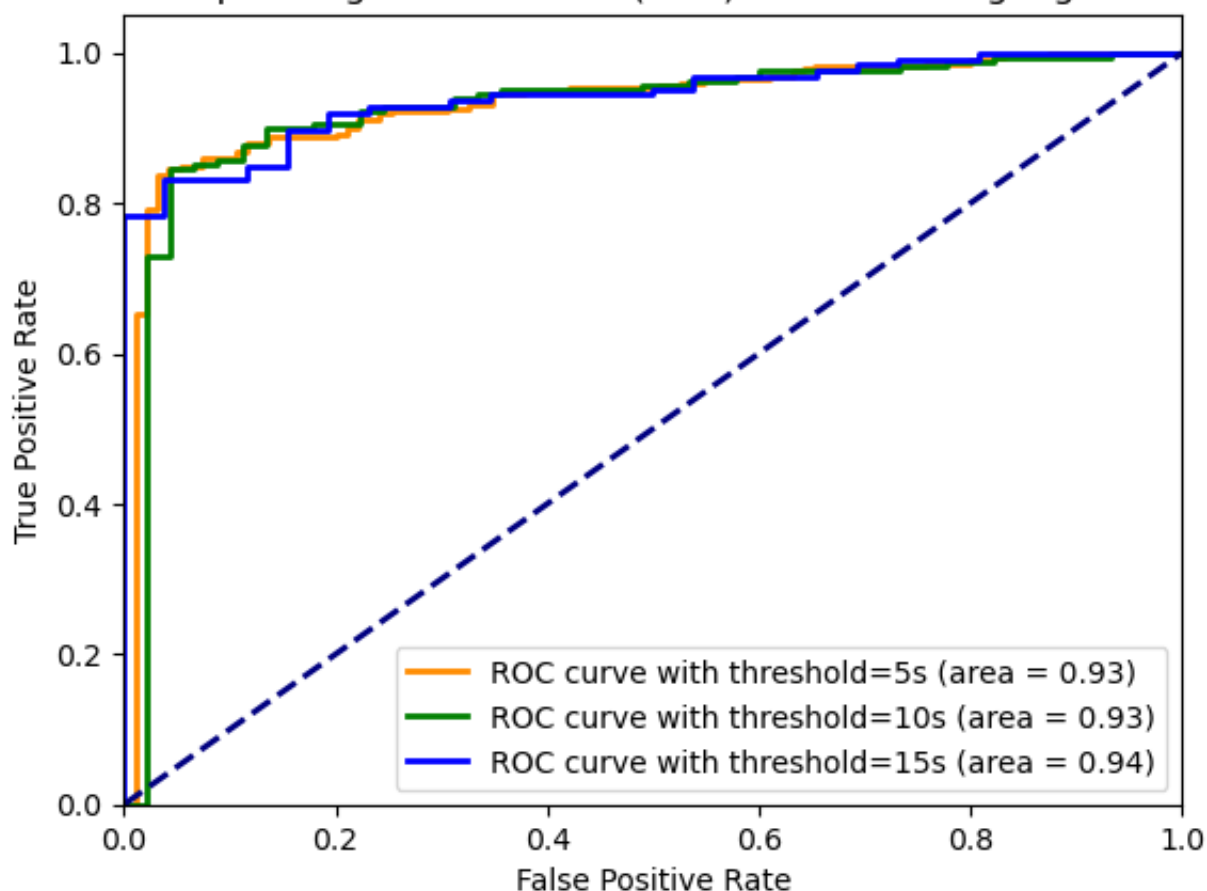
```
MAX_Score: -53.96791123100373 --- Predicted Language: french
MAX_Score: -45.22795106982991 --- Predicted Language: spanish
MAX_Score: -39.98285000412849 --- Predicted Language: spanish
MAX_Score: -50.546837601976776 --- Predicted Language: spanish
MAX_Score: -50.96863783804755 --- Predicted Language: spanish
MAX_Score: -50.03575007000474 --- Predicted Language: spanish
```

# RÉSULTATS OBTENUS :

## Pour la langue Française :

Nous avons obtenu de très bons résultats pour la langue espagnole en utilisant le modèle gmm512. En effet pour les segments de 5 s et de 10s l'auc est de 0.93 et pour les segments de 15s l'auc est 0.94. La courbe Roc résume ces résultats obtenus .

Receiver operating characteristic (ROC) for french language detection



---

# RÉSULTATS OBTENUS :

## Pour la langue française:

Les segments mal classifiés ont été prédits comme étant des segments de la langue russe, anglaise ou arabe .

```
MAX_Score: -51.28436489467996 --- Predicted Language: french
MAX_Score: -52.577125745286544 --- Predicted Language: russian
MAX_Score: -53.85706707118793 --- Predicted Language: russian
MAX_Score: -50.97290794030746 --- Predicted Language: french
MAX_Score: -29.256043317150315 --- Predicted Language: french
MAX_Score: -58.36093987671511 --- Predicted Language: arabic
MAX_Score: -58.04462924674877 --- Predicted Language: arabic
MAX_Score: -55.99805890490121 --- Predicted Language: russian
MAX_Score: -54.39474696492817 --- Predicted Language: english
MAX_Score: -55.25726994809528 --- Predicted Language: english
```

---

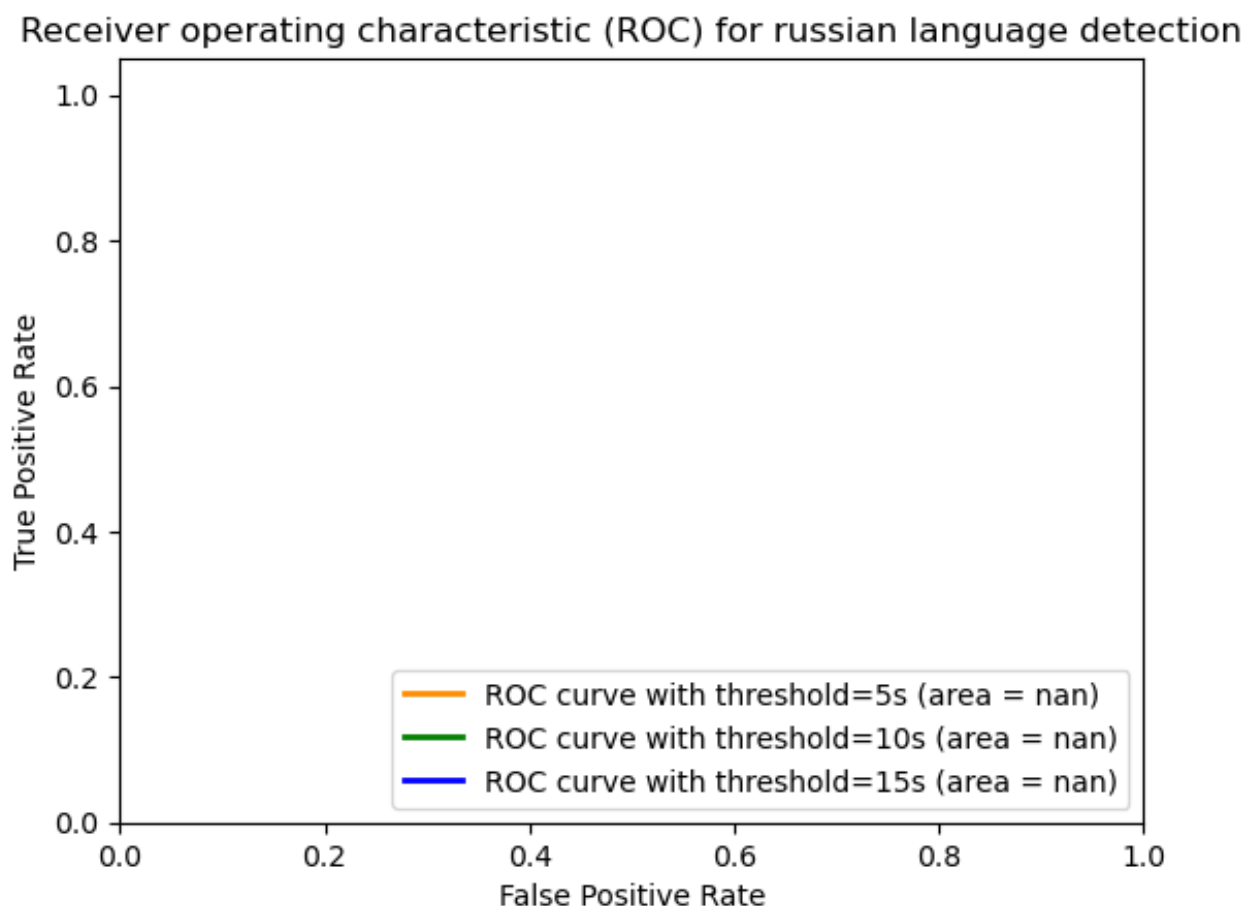
# RÉSULTATS OBTENUS :

## Pour la langue Japonaise :

Nous avons obtenu les meilleurs résultats pour la langue japonaise en utilisant le model gmm512. En effet pour les segments de 5 s de 10s et de 15s il n'y a aucune mal classification, tous les segments ont été prédits des segments Japonais.

La courbe Roc résume ces résultats obtenus .

(Dans ce cas on ne peut pas avoir les courbes puisqu'on a juste une seule classe prédite qui est le japonais donc on ne peut pas mesurer le taux de faux positifs).





---

# CONCLUSION

En guise de conclusion pour ce travail, on peut dire que nous avons obtenus des résultats en général satisfaisants. Les modèles de certaines langues étaient plus performants que d'autre. Cela revient à la qualité de la dataset. En effet pour le japonais et le russe où on a obtenu 100% de prédictions justes, les enregistrements ont été faits dans un même environnement ce qui rend le modèle moins sensible aux variations du milieu et donc il devient plus performant. Pour les autres langues, la dataset contenait du bruit et les enregistrements ont été fait dans des milieux différents ce qui a influencé la performance du modèle.

En outre, on a pu affirmer que plus la durée du segment à prédire est longue plus la prédiction est précise.

Finalement, ce travail peut être améliorer en travaillant sur la dataset.